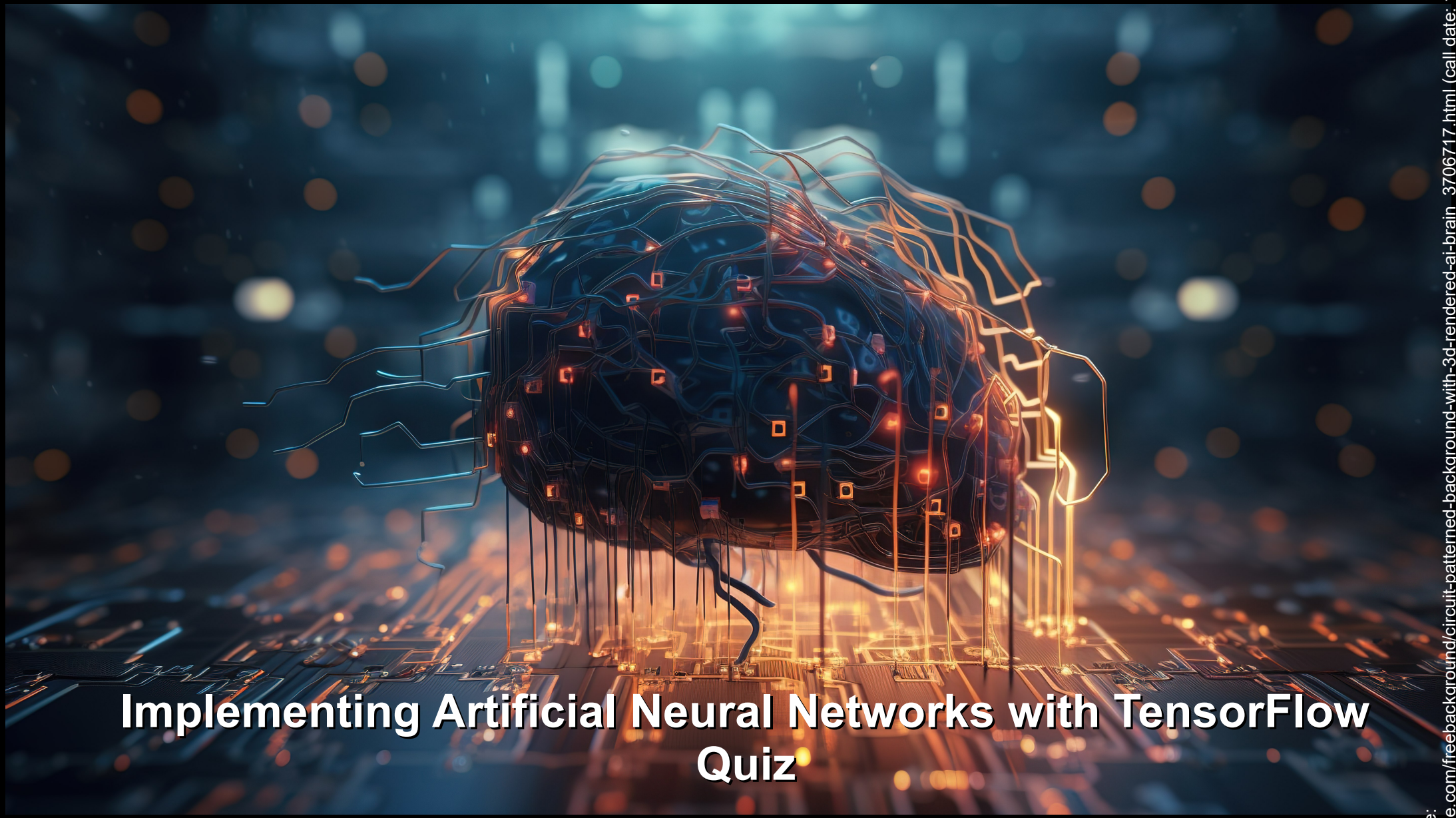


Implementing Artificial Neural Networks with TensorFlow Quiz

Tim Niklas Witte

Image source:
https://pngtree.com/freebackground/circuit-patterned-background-with-3d-rendered-ai-brain_3706717.html (call date: 14.12.23)

Please notice!

This quiz contains more questions than expected in the midterm.

It is not considered to be an “old midterm”.

I am not designing any questions for the midterm.

Some questions are on an advanced level for letting you intensely think about the lecture material. Such complicated questions are very unlikely to appear in the midterm in my opinion.

Task 1: Why Deep Learning?

Compared to “traditional machine learning methods, what are the main advantages of deep-learning-based approaches?

Feature Learning and Representation: Deep learning models automatically learn hierarchical representations of data. In traditional machine learning, feature engineering is often a manual and time-consuming process. Deep learning algorithms can automatically learn relevant features from raw data, reducing the need for extensive feature engineering.

Scalability: Deep learning models can handle large amounts of data efficiently, making them suitable for big data applications. As the size of the dataset increases, deep learning models can scale and often continue to improve in performance, whereas traditional methods may struggle to maintain accuracy with growing datasets.

Complexity of Relationships: Deep learning models can capture complex relationships and patterns in data. They excel at modeling intricate, non-linear relationships, which can be challenging for traditional models to capture without explicitly defined rules.

End-to-End Learning: Deep learning models can be trained end-to-end, allowing the entire system to be optimized simultaneously. This is in contrast to traditional machine learning pipelines, where different components may need to be individually fine-tuned.

Transfer Learning: Deep learning models trained on large datasets for one task can often be repurposed for related tasks with minimal adjustments. This is especially beneficial when labeled data for a specific task is limited, as pre-trained models can leverage knowledge gained from other domains.

Adaptability to Various Data Types: Deep learning models can handle a wide range of data types, including images, text, audio, and sequential data. Convolutional Neural Networks (CNNs) for images, Recurrent Neural Networks (RNNs) for sequential data, and Transformers for text are examples of deep learning architectures specialized for different data types.

Automatic Feature Extraction: Deep learning models automatically learn relevant features from raw data, eliminating the need for explicit feature extraction. This is particularly advantageous when dealing with unstructured data, such as images and text.

Continuous Improvement: Deep learning models often improve with more data and computing resources. As the field continues to evolve, new architectures and techniques are developed, contributing to ongoing advancements in model performance.

Task 2: Why not Deep Learning?

What are potential drawbacks of deep learning methods?

While deep learning methods have achieved remarkable success in various applications, they also come with certain drawbacks and challenges. Some of the potential drawbacks include:

1. **Data Requirements:** Deep learning models often require large amounts of labeled data for training. Acquiring and annotating such datasets can be expensive and time-consuming, making it challenging for applications with limited labeled data.

2. **Computational Resources:** Training deep learning models can be computationally intensive and may require specialized hardware such as GPUs or TPUs. This can pose challenges for smaller organizations or individuals with limited access to high-performance computing resources.

3. **Overfitting:** Deep learning models, especially complex ones with many parameters, are prone to overfitting, where the model performs well on the training data but fails to generalize effectively to new, unseen data. Regularization techniques and appropriate validation strategies are needed to mitigate this issue.

4. **Interpretability:** Deep learning models are often considered "black boxes" because it can be challenging to interpret how they arrive at specific decisions. Understanding the reasoning behind a deep learning model's predictions is crucial, especially in fields where interpretability is essential, such as healthcare and finance.

5. **Lack of Explainability:** The inner workings of deep learning models can be difficult to explain to non-experts. This lack of explainability can be a barrier to the adoption of deep learning in certain industries and applications where transparency and interpretability are critical.

6. **Training Time:** Training deep learning models can take a significant amount of time, especially for large datasets and complex architectures. This long training time can hinder the rapid development and experimentation required in certain applications.

7. **Need for Preprocessing:** Despite their ability to automatically learn features, deep learning models may still benefit from data preprocessing. Ensuring data quality, handling missing values, and normalizing data can be important for achieving optimal performance.

8. **Hyperparameter Tuning:** Deep learning models often have many hyperparameters that need to be tuned to achieve optimal performance. Finding the right combination of hyperparameters can be time-consuming and requires expertise.

9. **Vulnerability to Adversarial Attacks:** Deep learning models can be sensitive to small, carefully crafted perturbations in the input data, known as adversarial attacks. This vulnerability is a concern in security-critical applications.

10. **Resource-Intensive Inference:** Some deep learning models, especially large ones, can be resource-intensive during the inference phase. This can be a limitation in applications where real-time or low-latency predictions are required.

It's important to consider these drawbacks when deciding whether to use deep learning methods for a particular task. Depending on the application and the available resources, alternative machine learning approaches may be more suitable. Additionally, ongoing research is focused on addressing and mitigating these challenges to make deep learning more accessible and robust.

Task 3: Artificial vs. biological neural networks

Enumerate three differences between artificial and biological neural networks

Artificial Neural Networks (ANNs) and Biological Neural Networks (BNNs) share some similarities in their inspiration, but there are notable differences between them. Here are three key distinctions:

1. **Architectural Complexity:**
   - *Artificial Neural Networks (ANNs):* ANNs typically have a simpler and more structured architecture compared to biological neural networks. ANNs are designed with layers of interconnected nodes (neurons), each layer performing specific functions like input processing, feature extraction, and output generation. The architecture is often more regular and layered for ease of training and computation.

   - *Biological Neural Networks (BNNs):* The structure of biological neural networks is highly complex and interconnected. Neurons in the brain form intricate networks with massive parallelism and connectivity. The brain's architecture is not as neatly layered as in ANNs; instead, it consists of densely interconnected neurons with feedback loops and connections that span across various brain regions.

2. **Learning Mechanisms:**
   - *Artificial Neural Networks (ANNs):* Learning in ANNs is primarily achieved through supervised or unsupervised learning algorithms. Gradient descent and backpropagation are commonly used for adjusting the weights of connections between neurons to minimize the error between predicted and actual outputs. ANNs rely on mathematical optimization techniques to update model parameters during training.

   - *Biological Neural Networks (BNNs):* Learning in the brain is a highly dynamic and adaptive process. It involves complex mechanisms such as synaptic plasticity, where the strength of connections between neurons can change based on experience. Biological systems exhibit a wide range of learning mechanisms, including Hebbian learning, spike-timing-dependent plasticity (STDP), and various forms of reinforcement learning.

3. **Energy Efficiency and Parallelism:**
   - *Artificial Neural Networks (ANNs):* While modern hardware accelerators like GPUs are efficient for training ANNs, the energy consumption can be relatively high compared to biological systems. ANN training often involves large-scale matrix operations and may require significant computational resources.

   - *Biological Neural Networks (BNNs):* The brain is highly energy-efficient and operates in a massively parallel manner. Neurons in the brain consume relatively low amounts of energy, and the brain's architecture allows for efficient parallel processing. This efficiency is partly due to the brain's ability to activate only relevant regions and neurons for specific tasks, reducing unnecessary energy consumption.

These differences highlight that while artificial neural networks are inspired by the structure and function of biological neural networks, they are simplified abstractions designed for computational efficiency and tractability. Biological neural networks, on the other hand, remain a vastly more complex and adaptable system with unique learning mechanisms and energy-efficient parallel processing capabilities.

Task 4: Backpropagation and the brain*

Based on your knowledge derive two reasons why the backpropagation algorithm can not be applied in the brain i.e. it is biological implausible.

While backpropagation is a widely used algorithm for training artificial neural networks (ANNs), it faces challenges when considering its implementation in the biological neural networks (BNNs) of the brain. Here are two reasons why backpropagation is considered biologically implausible:

1. **Error Signal Propagation:**
   - *Backpropagation in ANNs:* In artificial neural networks, backpropagation relies on the availability of a clear and instantaneous error signal, which is the difference between the predicted output and the actual target. This error signal is used to adjust the weights of the connections between neurons in the network. The backward pass of backpropagation involves propagating this error signal from the output layer back through the network to adjust the weights.

   - *Biological Neural Networks (BNNs):* In the brain, the concept of an explicit and instantaneous error signal is challenging. Neurons in the brain do not have direct access to a global error signal for each specific task. Learning in biological systems is believed to involve more complex mechanisms, including local learning rules like Hebbian plasticity and reward-based learning. The brain may not have a centralized, error-specific signal that is required by backpropagation.

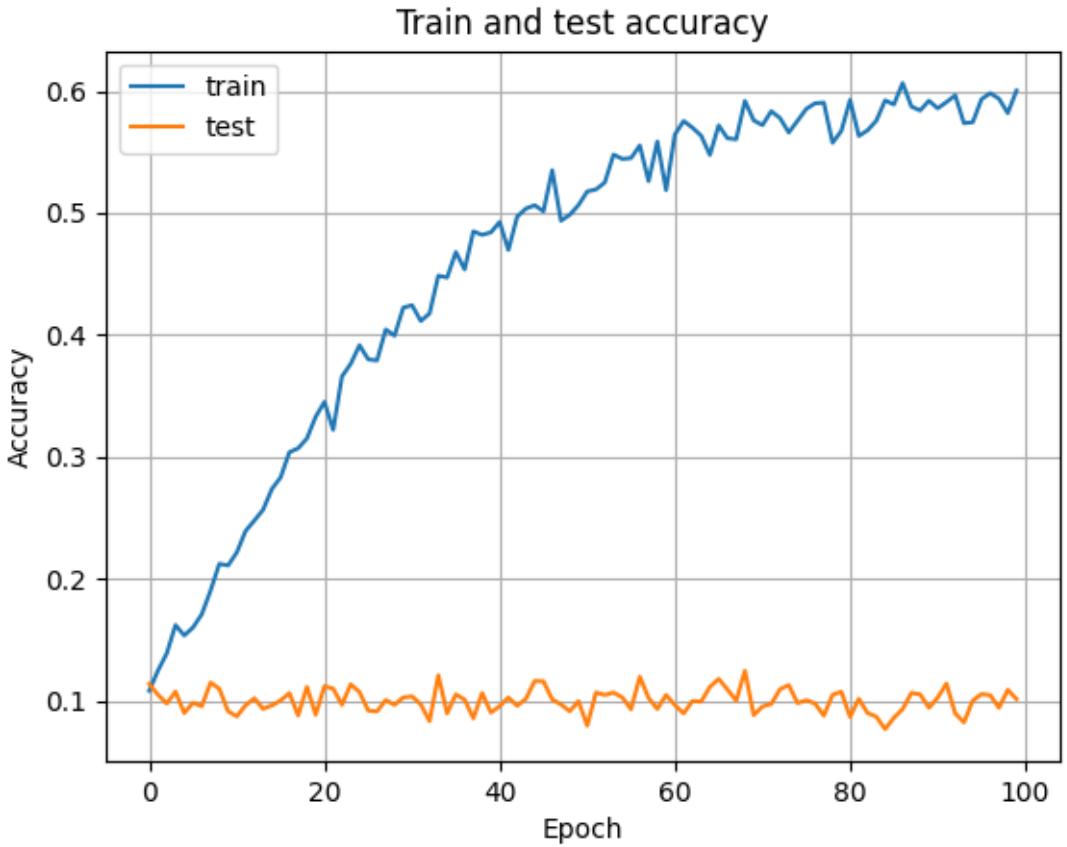
2. **Credit Assignment Problem:**
   - *Backpropagation in ANNs:* Backpropagation assumes a clear and direct assignment of credit or blame to individual neurons in the network for their contribution to the error. During the backward pass, the algorithm assigns gradients to each connection, indicating how much each connection contributed to the error. This credit assignment process is crucial for updating the weights effectively.

   - *Biological Neural Networks (BNNs):* The credit assignment problem is more complex in biological systems. The brain operates in a massively parallel and distributed manner, making it challenging to attribute specific errors to individual neurons or synapses. Biological learning mechanisms, such as synaptic plasticity and neuromodulation, involve more subtle and distributed changes, and the brain likely uses a combination of local and global signals for credit assignment. The distributed and parallel nature of credit assignment in the brain contrasts with the more centralized and precise assignment assumed by backpropagation.

In summary, the biological implausibility of backpropagation in the brain arises from the lack of an explicit and instantaneous error signal, as well as the challenge of credit assignment in the highly parallel and distributed architecture of biological neural networks. Researchers exploring biologically inspired learning algorithms aim to develop approaches that better align with the complexity and characteristics of the brain's learning mechanisms.

Task 5: Train and test accuracy

Given the following plot of the train and test accuracy of a CNN model trained on the MNIST dataset :

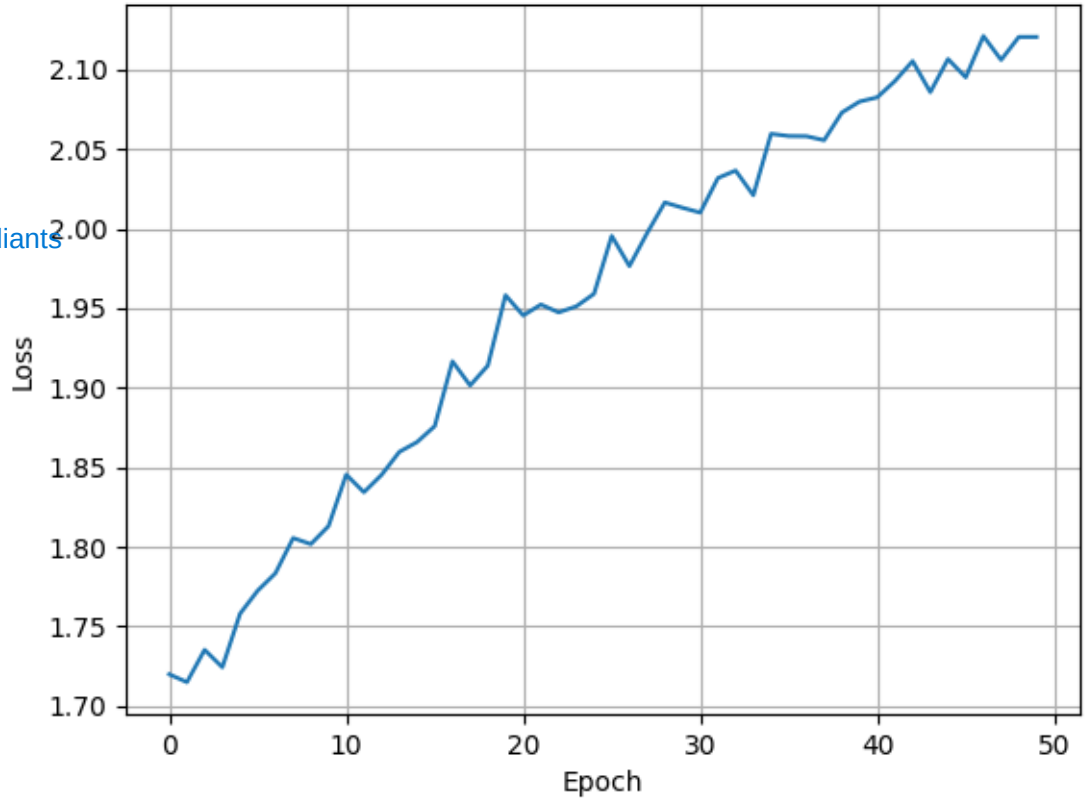


The train accuracy is increasing while the test accuracy is almost constant. Name two possible reasons why this can happen.
(Hint: Programming errors)

Task 6: A tiny detail but sooo important

A student has implemented his own neural network including the backpropagation algorithm (homework 1). Now he trained his network on the MNIST dataset (CCE loss). The train loss per epoch is displayed in the following plot:

not unsing negative gradiants



Which typical *tiny* programming error result into a loss increase instead of loss decrease?

Task 7: Bias

Given the statement:

“I don't need the bias!”

Explain why a neural network without a bias (parameter) performs worse compared to a neural network with bias (parameter)

Task 8: Mean squared error

a) Why we use the mean squared error as a loss function for regression?

b) Can I use this loss function without the square?

c) Based on your knowledge derive one problem of the MSE loss function.

Task 9: Squared error loss?

Given the MSE loss function and the squared error (SE) loss function

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=0}^{N}(t_i - \hat{y}_i)^2$$

$$\mathcal{L}_{\text{SE}} = \sum_{i=0}^{N}(t_i - \hat{y}_i)^2$$

You want to train a neural network on a regression task.
Which of these two loss function you choose and why?

Task 10: tanh vs ReLU

Compare the tanh and ReLU activations functions in terms their advantages and disadvantages (one each)

Task 11: Activations

Why is beneficial to have the activations of a layer in the range of [-1, 1] instead of [0, 255]?

Task 12: Generator

Given the following code of a generator function which creates data for a TensorFlow dataset

```
def generate_data():
    while True:
        yield np.random.normal(size=(28,28,1))

if __name__ == "__main__":
    train_ds = tf.data.Dataset.from_generator(
                    generate_data,
                    output_signature=(
                        tf.TensorSpec(shape=(28,28,1), dtype=tf.uint8),
                    )
                )
```

This code contains two programming errors (no syntax). Find these two.

Task 13: Preprocessing pipeline

Given the following code of a preprocessing pipeline for the MNIST dataset:

```
def prepare_data(dataset):

    dataset = dataset.map(lambda img, target: (tf.reshape(img, (-1,)), target))
    dataset = dataset.map(lambda img, target: (tf.cast(img, tf.float32), target) )
    dataset = dataset.cache()
    dataset = dataset.map(lambda img, target: (img, tf.one_hot(target, depth=10)))

    dataset = dataset.batch(32)
    dataset = dataset.shuffle(1350)

    dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)

    return dataset
```

The neural network contains only dense layers.
The code contains two errors which result into a bad performance (accuracy and loss) of the neural network. Find these errors!

Task 14: TF datasets

Name the key difference between *dataset.map* and *dataset.apply*?

Task 15: Memory leaks in TF?

Given the following code block belonging to a model:

```
@tf.function
def train_step(self, data, optimiser):

    for x, target in data:

        with tf.GradientTape() as tape:
            pred = self.model(x)
            loss = self.loss_function(target, pred)

        gradients = tape.gradient(loss, self.model.trainable_variables)
        optimiser.apply_gradients(zip(gradients, self.model.trainable_variables))
```

When training the model on the MNIST dataset, almost all GPU memory is being used and an epoch takes almost 10x longer than normal (error fixed). Where is this error?

Task 16: Eager vs. graph execution

What is the difference between eager and graph execution?
(two sentences)

Task 17: TensorFlow model class

Given the following code of tensorflow model

```
import tensorflow as tf

class Classifier(tf.keras.Model):

    def __init__(self):

        super(Classifier, self).__init__()

        self.layer_list = [
            tf.keras.layers.Dense(16, activation="tanh"),
            tf.keras.layers.Dense(10, activation=tf.nn.softmax)
        ]
```

The Classifier class has no layers attribute. Why does the following code run without an error?

```
classifier = Classifier()
classifier.layers
```

Task 18: Non-convex optimization

What is a non-convex optimization problem? (one sentence)
Name one example

Task 19: Overfitting

What is overfitting?

Enumerate reasons causing it.

Why is this bad?

How to avoid it?

Task 20: Underfitting

What is underfitting?

Enumerate reasons causing it.

Why is this bad?

How to avoid it?

Task 21: Error surface

Draw two error surfaces (2D: x-y plane) in which stochastic gradient descent performs bad.

and explain why

Task 22: Optimizers

Explain the key idea of AdaGrad

What is the difference to RMSprob?

Did AdaGrad uses momentum?

What is the main issue with AdaGrad?

RMSprob performs better than AdaGrad. True or false?

Task 23: Escape local minima

Name two techniques enable escape of local minima during training and explain why.

Task 24: Single item SGD vs. full batch SGD

Name one advantage and one disadvantage of single item and full batch SGD

Task 25: Batch size

A higher batch size helps with noisy data. True or false?
Explain your decision

Which effect has the batch size on the speed of convergence?

Task 26: CNNs performance

Why CNNs perform better in image classification compared to networks based on only dense layers?
(assume same number of parameters)

Task 27: CNNs overfitting

Can the convolution operation in a CNN be considered as a method against overfitting?

Task 28: CNNs: feature maps = activation maps?

What is the difference between a feature map and an activation map?

Task 29: CNNs: Number of parameters

Given an input image of shape (28, 28, 1)

And

Number of parameter
Output shapes

Task 30: CNNs: Value in an activation map

What does a high value in the activation map indicate?

Task 31: CNN fail I

What is the problem with this network for image classification?
(ignore the image size)

```
tf.keras.layers.Conv2D(filters=2, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='same')
tf.keras.layers.Conv2D(filters=4, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='same')
MaxPooling2D( pool_size=(2, 2), strides=(1,1))

tf.keras.layers.Conv2D(filters=8, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='same')
tf.keras.layers.Conv2D(filters=16, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='same')
MaxPooling2D( pool_size=(2, 2), strides=(1,1))

tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='same')
tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='same')

tf.keras.layers.Flatten()
tf.keras.layers.Dense(num_classes, activation="softmax")
```

Task 32: CNNs fail II

What is the problem with this network for image classification?
Given an image of size 32x32 pixiels

```
tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='valid')
tf.keras.layers.Conv2D(filters=40, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='valid')
tf.keras.layers.Conv2D(filters=50, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='valid')
MaxPooling2D( pool_size=(2, 2), strides=(2,2))

tf.keras.layers.Conv2D(filters=60, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='valid')
tf.keras.layers.Conv2D(filters=70, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='valid')
tf.keras.layers.Conv2D(filters=80, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='valid')
MaxPooling2D( pool_size=(2, 2), strides=(2,2))

tf.keras.layers.Conv2D(filters=60, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='valid')
tf.keras.layers.Conv2D(filters=70, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='valid')
tf.keras.layers.Conv2D(filters=80, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='valid')

tf.keras.layers.GlobalAveragePooling2D()
tf.keras.layers.Dense(num_classes, activation="softmax")
```

Task 33: CNNs: Block

What is a block in a CNN?

Task 34: CNN design

Why is the feature map size in a CNN progressively decreased?

Task 35: Inference fail

Assume that in the file *./digit.png* is an image of MNIST digit stored and a CNN model was already trained on the MNIST dataset.

You want to do inference i.e. passing this image to the model to get prediction. In the below code the MNIST digit is being loaded and this CNN model is being created. Besides, it's parameter (weights + biases) are loaded. However, line in which you pass the image to the model causes an error. Why?

```
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np

image = plt.imread('./MNIST_digit.png')
# image.shape = (28,28,1)

classifier = # creating the model and loading its parameters correctly :)

prediction = classifier(image) # causes an error

digit = np.argmax(prediction)
```

Task 36: CNNs Pooling I

There are the following pooling operations:
- MaxPooling
- AveragePooling

Which of those two is being preferred and why?

Task 37: CNNs Pooling II

What is the problem with this pooling operation?

```
MaxPooling2D(pool_size=(2, 2), strides=(1,1))
```

Task 38: CNN output shape

What is output shape after passing a tensor of the shape (32, 1920, 1080, 3) through the following layers:

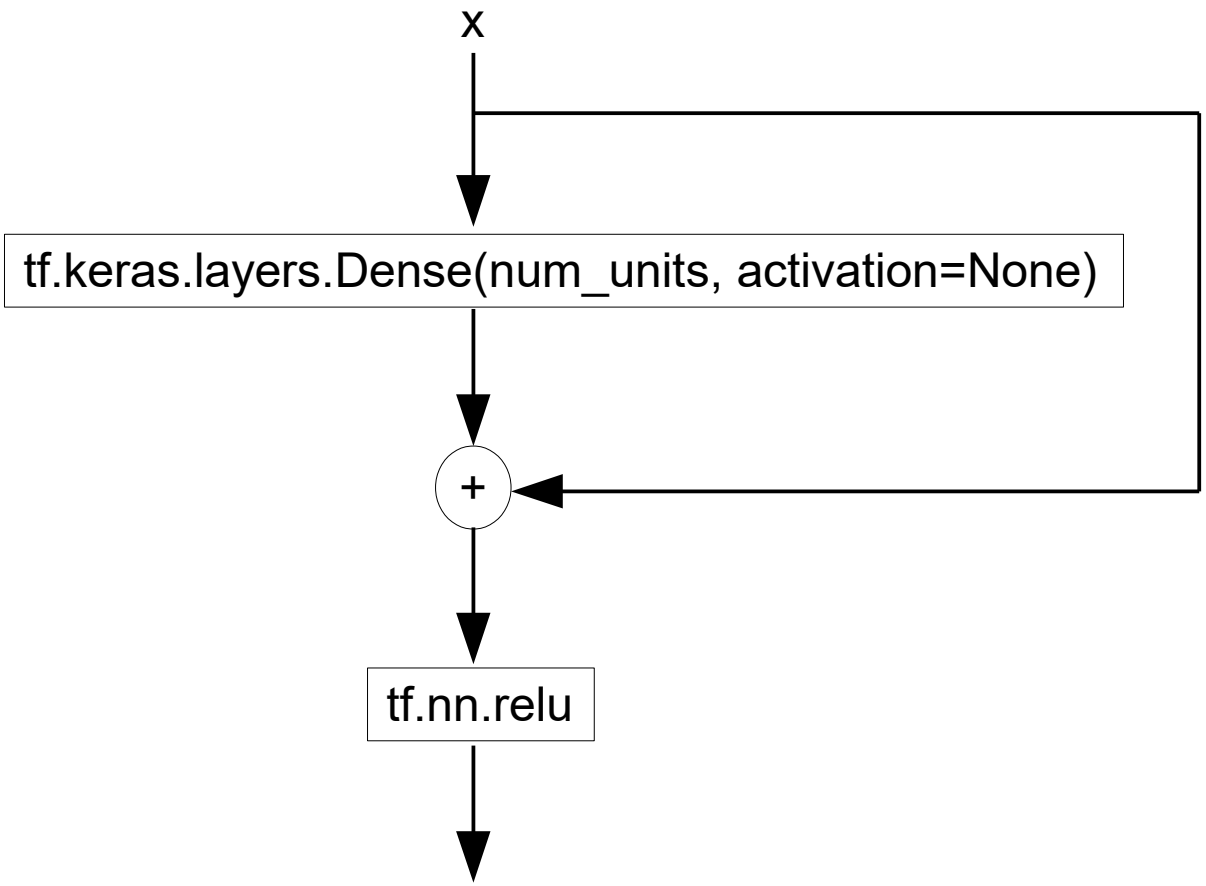
```
tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), activation="tanh", padding='valid')

tf.keras.layers.GlobalAveragePooling2D()
```

Task 39: CNNs receptive field

What is the key idea of the receptive field?
How does it influence the design of a CNN?

Task 40: Residual block design fail

Given the following design of a residual block.
What is the problem with this design?



Task 41: SGD modification

Given the following figure from the ResNet paper:

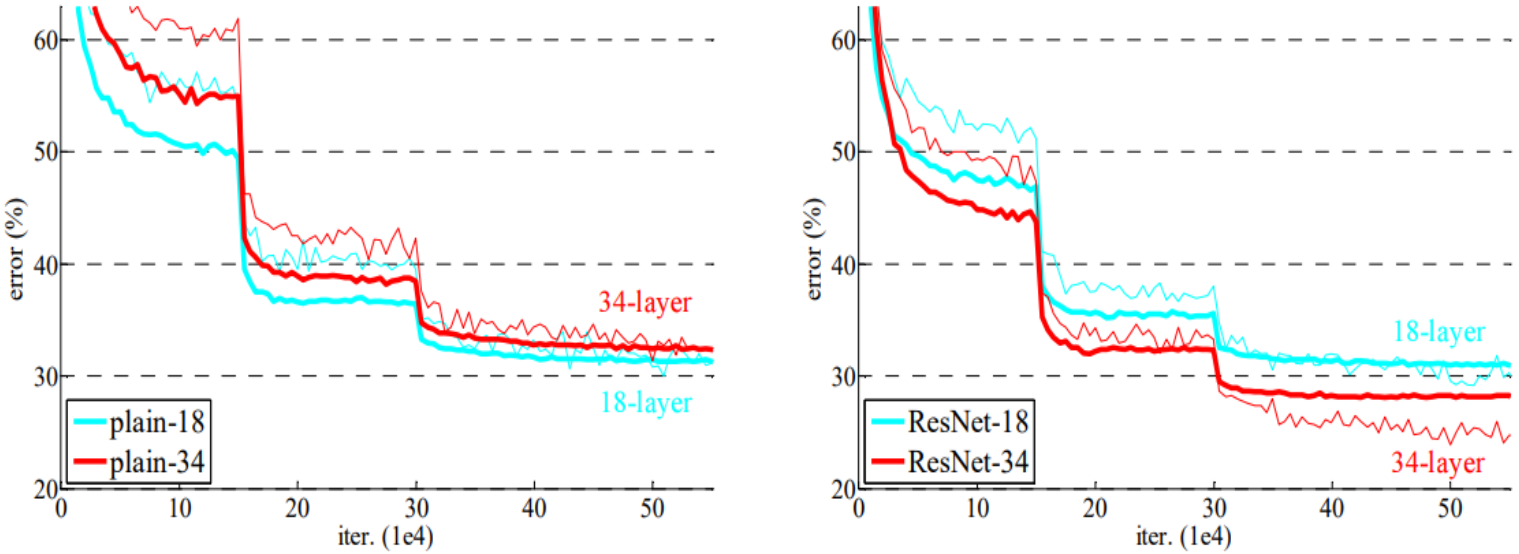


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

The training was performed with a modified stochastic gradient descent (only SGD, no ADAM, RMSprob etc.).
Derive this modification of SGD which causes this “spontaneous collapses” of the error?

Image source: He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.

Task 42: CNNs bottleneck

What is a bottleneck in the CNN architecture?

What is its purpose in the ResNet and DenseNet architecture?

Task 43: CNNs Pooling III

Explain with an example why it is always a good idea to set pool_size = stride in a MaxPooling2D layer.

Task 44: DenseNet modified

Assume in a DenseNet architecture the add operation is performed instead of the concatenate operation. Derive resulting problems based on this modification.

Task 45: Hyperparameters

Enumerate hyperparameters which can influence the model's performance.

Task 46: Fully connected model fail

What is the problem with the model consisting of the layers shown below for MNIST digit classification? Explain why.

```
tf.keras.layers.Dense(20, activation=None)
tf.keras.layers.Dense(30, activation=None)
tf.keras.layers.Dense(50, activation=None)
tf.keras.layers.Dense(10, activation="softmax")
```

Task 47: Deeper or wider?

What is more important the depth of the model (number of layer) or the layer sizes? Explain why.

Task 48: Weight updates

What is the major issue with the exploding gradient problem in terms of updating the weights?

Task 49: Failed regularization

Given the following code of a model for MNST digit classification
Why is the weight regularization of the Conv2D layer not working?

```
import tensorflow as tf

class Classifier(tf.keras.Model):

    def __init__(self):

        super(Classifier, self).__init__()
        num_classes = 10

        self.layer_list = [
            tf.keras.layers.Conv2D(filters=40, kernel_size=(4,4),
                    strides=(4,4), activation="tanh",
                    padding='same', kernel_regularizer_tf.keras.regularizers.L2(0.01)),

              tf.keras.layers.Flatten()
               tf.keras.layers.Dense(num_classes, activation="softmax")
        ]

    @tf.function
    def call(self, x):
        for layer in self.layer_list:
            x = layer(x)
        return x

    @tf.function
    def train_step(self, image, target):

        with tf.GradientTape() as tape:
            prediction = self(image)
            loss = self.loss_function(target, prediction)

        gradients = tape.gradient(loss, self.trainable_variables)
        self.optimizer.apply_gradients(zip(gradients, self.trainable_variables))
```

Task 50: Favourite loss landscape

How to favour smooth and broader local optima over sharp local optima?
Hint: Hyperparameter

Task 51: Better generalizability

Why does broader and smoother local optima generalize better than sharp local optima?

Task 52: Normalization

Assume you have not normalized your input data. Which problem is more likely to occur during backpropagation and why?

Task 53: Initialization

Why is initializing the model weights with zeros a bad idea?

Task 54: Backpropagation

In the backpropagation algorithm when the delta value $\delta^{(L)}$ is computed for the current layer (L) the weights for the previous layer (L+1) are considered for this computation. These weights have been changed during in the previous step.

When computing $\delta^{(L)}$ are the old or the changed weights being used? Explain what will happened otherwise if layer applying this to a deep network.

Task 55: BatchNormalization failed

Given the following code of a model for MNST digit classification
Find the two programming mistakes involving the BatchNormalization

```
import tensorflow as tf

class Classifier(tf.keras.Model):

    def __init__(self):

        super(Classifier, self).__init__()

        self.bm_layer = tf.keras.layers.BatchNormalization()
        self.layer_list = [
                tf.keras.layers.Dense(10, activation="tanh")
                self.bm_layer,
                tf.keras.layers.Dense(10, activation="tanh")
                self.bm_layer,
               tf.keras.layers.Dense(10, activation="softmax")
        ]

    @tf.function
    def call(self, x):
        for layer in self.layer_list:
            x = layer(x)
        return x

    @tf.function
    def train_step(self, image, target):

        with tf.GradientTape() as tape:
            prediction = self(image)
            loss = self.loss_function(target, prediction)

        gradients = tape.gradient(loss, self.trainable_variables)
        self.optimizer.apply_gradients(zip(gradients, self.trainable_variables))
```

Task 56: ReLU vs. tanh

Which is these two activation functions is more likely to cause vanishing gradients? Explain why